



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Modelos Avanzados de Bases de Datos.
Funcionalidad 1

Bases de datos Orientadas a Objetos y
Bases de Datos Objeto-Relacionales

Alejandro Alberca Manzaneque

Jesús Galvez Díaz-Tendero

Índice

Parte Bases de datos orientadas a objetos:

1.- Introducción	3
2.- Conceptos relacionados con las bases de datos orientadas a objetos	3
3.- Origen de las Bases de Datos Orientadas a Objetos	3
4.- Características de las Bases de Datos Orientadas a Objetos y diferencias de éstas con respecto a las relacionales	5
5.- Manifiesto Malcolm Atkinson: características de un BDOO	6
6.- Ventajas e inconvenientes de las BDOO	7
7.- ODMG: el estándar de facto para modelos de objetos	8

Parte Bases de datos objeto-relacionales:

1.- Introducción a las Bases de datos Objeto-Relacionales	10
2.- Características de las Bases de datos Objeto-Relacionales	10
3.- Tipos de Datos definidos por el Usuario	11
4.- Herencia de tipos	13
5.- POSTGRES	14
6.- SQL 99	14

1.- Introducción

Las bases de datos orientadas a objetos (BDOO) son aquellas cuyo modelo de datos está orientado a objetos y almacenan y recuperan objetos en los que se almacena estado y comportamiento. Su origen se debe a que en los modelos clásicos de datos existen problemas para representar cierta información, puesto que aunque permiten representar gran cantidad de datos, las operaciones que se pueden realizar con ellos son bastante simples.

Las clases utilizadas en un determinado lenguaje de programación orientado a objetos son las mismas clases que serán utilizadas en una BDOO; de tal manera, que no es necesaria una transformación del modelo de objetos para ser utilizado por un SGBDOO. De forma contraria, el modelo relacional requiere abstraerse lo suficiente como para adaptar los objetos del mundo real a tablas.

Las bases de datos orientadas a objetos surgen para evitar los problemas que surgen al tratar de representar cierta información, aprovechar las ventajas del paradigma orientado a objetos en el campo de las bases de datos y para evitar transformaciones entre modelos de datos (usar el mismo modelo de objetos).

2.- Conceptos relacionados con las bases de datos orientadas a objetos

En este apartado se explican los conceptos relacionados con las BDOO:

- ❑ **Base de datos orientada a objetos (BDOO):** una colección persistente y compatible de objetos definida por un modelo de datos orientado a objetos.
- ❑ **Modelo de datos orientado a objetos:** Un modelo de datos que captura la semántica de los objetos soportados en la programación orientada a objetos.
- ❑ **Sistema Gestor de Bases de Datos Orientadas a Objetos (SGBDOO):** El gestor de una base de datos orientada a objetos.

3.- Origen de las Bases de Datos Orientadas a Objetos

El origen de las BDOO se encuentra básicamente en las siguientes razones:

- ❑ la existencia de problemas para representar cierta información y modelar ciertos aspectos del ‘mundo real’, puesto que los modelos clásicos permiten representar gran cantidad de datos, pero las operaciones y representaciones que se pueden realizar sobre ellos son bastante simples.

- ❑ El paso del modelo de objetos al modelo relacional genera dificultades que en el caso de las BDOO no surgen ya que el modelo es el mismo.

Por lo tanto, las bases de datos orientadas a objetos surgen básicamente para tratar de paliar las deficiencias de los modelos anteriores y para proporcionar eficiencia y sencillez a las aplicaciones.

Las debilidades y limitaciones de los SGBDR son:

- ❑ Pobre representación de las entidades del ‘mundo real’.
- ❑ Sobrecarga y poca riqueza semánticas.
- ❑ Soporte inadecuado para las restricciones de integridad y empresariales
- ❑ Estructura de datos homogénea
- ❑ Operaciones limitadas
- ❑ Dificultades para gestionar las consultas recursivas
- ❑ Desadaptación de impedancias
- ❑ Problemas asociados a la concurrencia, cambios en los esquemas y el inadecuado acceso navegacional.
- ❑ No ofrecen soporte para tipos definidos por el usuario (sólo dominios)

Mientras que las necesidades de las aplicaciones actuales con respecto a las bases de datos son:

- ❑ Soporte para objetos complejos y datos multimedia
- ❑ Identificadores únicos
- ❑ Soporte a referencias e interrelaciones
- ❑ Manipulación navegacional y de conjunto de registros
- ❑ Jerarquías de objetos o tipos y herencia
- ❑ Integración de los datos con sus procedimientos asociados
- ❑ Modelos extensibles mediante tipos de datos definidos por el usuario
- ❑ Gestión de versiones
- ❑ Facilidades de evolución
- ❑ Transacciones de larga duración
- ❑ Interconexión e interoperabilidad

Debido a las limitaciones anteriormente expuestas, el uso de BDOO es más ventajoso si se presenta en alguno de los siguientes escenarios:

- ❑ Un gran número de tipos de datos diferentes
- ❑ Un gran número de relaciones entre los objetos
- ❑ Objetos con comportamientos complejos

Se puede encontrar este tipo de complejidad acerca de tipos de datos, relaciones entre objetos y comportamiento de los objetos principalmente en aplicaciones de ingeniería, manufacturación, simulaciones, automatización de oficina y en numerosos sistemas de información. No obstante, las BDOO no están restringidas a estas áreas. Ya que al ofrecer la misma funcionalidad que su precursoras relacionales, el resto de campos de aplicación tiene la posibilidad de aprovechar completamente la potencia que las BDOO ofrecen para modelar situaciones del mundo real.

4.- Características de las Bases de Datos Orientadas a Objetos y diferencias de éstas con respecto a las relacionales.

Mientras que en una BDR los datos a almacenar se almacenan representados en tablas en un BDOO los datos se almacenan como objetos. Un objeto en BDOO como en POO es una entidad identificable unívocamente que describe tanto el estado como el comportamiento de una entidad del 'mundo real'. El estado de un objeto es descrito mediante atributos mientras que su comportamiento es definido mediante métodos.

Los características asociadas a las BDOO son:

- ❑ **Objetos:** cada entidad del mundo real se modela como un objeto.
- ❑ La forma de identificar objetos es mediante un **identificador de objetos** (OID, Object Identifier), único para cada objeto. Generalmente este identificador no es accesible ni modificable para el usuario (modo de aumentar la integridad de entidades y la integridad referencial). Los OID son independientes del contenido. Es decir, si un objeto cambia los valores de atributos, sigue siendo el mismo objeto con el mismo OID. Si dos objetos tienen el mismo estado pero diferentes OID, son equivalentes pero tienen identidades diferentes.
- ❑ **Encapsulamiento:** cada objeto contiene y define procedimientos (métodos) y la interfaz mediante la cual se puede acceder a él y otros objetos pueden manipularlo. La mayoría de los SGBDOO permite el acceso directo a los atributos incluyendo operaciones definidas por el propio SGBDOO las cuales leen y modifican los atributos para evitar que el usuario

tenga que implementar una cantidad considerable de métodos cuyo único propósito sea el de leer y escribir los atributos de un objeto. Generalmente, los SGBDOO permiten al usuario especificar qué atributos y métodos son visibles en la interfaz del objeto y pueden invocarse desde afuera.

- ❑ **Otros conceptos** utilizados de la misma manera que en la POO son:
 - **Clases**
 - **Herencia simple, múltiple y repetida.**
 - **Polimorfismo de operación, de inclusión y paramétrico; ligadura tardía** (late binding); **sobrecarga** (overloading) y **suplantación o anulación** (overriding).
 - **Objetos complejos**

5.- Manifiesto Malcolm Atkinson: características de un BDOO

En 1989 se hizo el Manifiesto de los sistemas de base de datos orientados a objetos el cual propuso trece características obligatorias para un SGBDOO y cuatro opcionales. Las trece características obligatorias estaban basadas en dos criterios: debía tratarse de un sistema orientado a objetos y un SGBD.

Características obligatorias de orientación a objetos:

- 1) Deben soportarse objetos complejos
- 2) Deben soportarse mecanismos de identidad de los objetos
- 3) Debe soportarse la encapsulación
- 4) Deben soportarse los tipos o clases
- 5) Los tipos o clases deben ser capaces de heredar de sus ancestros
- 6) Debe soportarse el enlace dinámico
- 7) El DML debe ser computacionalmente complejo
- 8) El conjunto de todos los tipos de datos debe ser ampliable

Características obligatorias de SGBD:

- 9) Debe proporcionarse persistencia a los datos
- 10) El SGBD debe ser capaz de gestionar bases de datos de muy gran tamaño
- 11) El SGBD debe soportar a usuarios concurrentes
- 12) El SGBD debe ser capaz de recuperarse de fallos hardware y software
- 13) El SGBD debe proporcionar una forma simple de consultar los datos.

Características opcionales:

- 1) Herencia múltiple
- 2) Comprobación de tipos e inferencia de tipos
- 3) Sistema de base de datos distribuido
- 4) Soporte de versiones

6.- Ventajas e inconvenientes de las BDOO

Aunque los SGBDOO pueden proporcionar soluciones apropiadas para muchos tipos de aplicaciones avanzadas de bases de datos, también tienen sus desventajas.

Las ventajas de un SGBDOO son:

- ❑ **Mayor capacidad de modelado.** El modelado de datos orientado a objetos permite modelar el ‘mundo real’ de una manera mucho más fiel. Esto se debe a:
 - un objeto permite encapsular tanto un estado como un comportamiento
 - un objeto puede almacenar todas las relaciones que tenga con otros objetos
 - los objetos pueden agruparse para formar objetos complejos (herencia).
- ❑ **Ampliabilidad.** Esto se debe a:
 - Se pueden construir nuevos tipos de datos a partir de los ya existentes.
 - Agrupación de propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
 - Reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.
- ❑ **Lenguaje de consulta más expresivo.** El acceso navegacional desde un objeto al siguiente es la forma más común de acceso a datos en un SGBDOO. Mientras que SQL utiliza el acceso asociativo. El acceso navegacional es más adecuado para gestionar operaciones como los despieces, consultas recursivas, etc.
- ❑ **Adecuación a las aplicaciones avanzadas de base de datos.** Hay muchas áreas en las que los SGBD tradicionales no han tenido excesivo éxito como el CAD, CASE, OIS, sistemas multimedia, etc. en los que las capacidades de modelado de los SGBDOO han hecho que esos sistemas sí resulten efectivos para este tipo de aplicaciones.
- ❑ **Mayores prestaciones.** Los SGBDOO proporcionan mejoras significativas de rendimiento con respecto a los SGBD relacionales. Aunque hay autores que han argumentado que los bancos de prueba usados están dirigidos a aplicaciones de ingeniería donde los SGBDOO son más adecuados. También está demostrado que los SGBDR tienen un rendimiento

mejor que los SGBDOO en las aplicaciones tradicionales de bases de datos como el procesamiento de transacciones en línea (OLTP).

Los inconvenientes de un SGBDOO son:

- ❑ **Carencia de un modelo de datos universal.** No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen una base teórica.
- ❑ **Carencia de experiencia.** Todavía no se dispone del nivel de experiencia del que se dispone para los sistemas tradicionales.
- ❑ **Carencia de estándares.** Existe una carencia de estándares general para los SGBDOO.
- ❑ **Competencia.** Con respecto a los SGBDR y los SGBDOR. Estos productos tienen una experiencia de uso considerable. SQL es un estándar aprobado y ODBC es un estándar de facto. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.
- ❑ **La optimización de consultas compromete la encapsulación.** La optimización de consultas requiere una comprensión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.
- ❑ El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.

7.- ODMG: el estándar de facto para modelos de objetos

ODMG es un grupo de representantes de la industria de bases de datos el cual fue concebido en el verano de 1991 con el objetivo de definir estándares para los SGBDOO. Uno de sus estándares, el cual lleva el mismo nombre del grupo (ODMG), es el del modelo para la semántica de los objetos de una base de datos. El modelo de objetos ODMG es un superconjunto del modelo de objetos de OMG, que permite portar tanto los diseños como las implementaciones entre diversos sistemas compatibles.

La última versión del estándar, ODMG 3.0, propone los siguientes componentes principales de la arquitectura ODMG para un SGBDOO:

- ❑ Modelo de objetos
- ❑ Lenguaje de definición de objetos (ODL, Object Definition Language)
- ❑ Lenguaje de consulta de objetos (OQL, Object Query Language)
- ❑ Conexión con los lenguajes C++, Smalltalk y Java (al menos)

El modelo de objetos ODMG permite que tanto los diseños como las implementaciones, sean portables entre los sistemas que lo soportan.

ODL es un lenguaje para definir la especificación de los tipos de objetos para sistemas compatibles con ODMG. ODL es el equivalente de DDL (Data Definition Language o lenguaje de definición de datos) de los SGBD tradicionales. Define los atributos y las relaciones entre tipos y especifica la signatura de las operaciones. Su principal objetivo es el de facilitar la portabilidad de los esquemas entre sistemas compatibles al mismo tiempo que proporciona interoperabilidad entre distintos SGBD. La sintaxis de ODL extiende el lenguaje de definición de interfaces (IDL) de la arquitectura CORBA (Common Object Request Broker Architecture).

OQL es un lenguaje declarativo del tipo de SQL que permite realizar consultas sobre bases de datos orientadas a objetos, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras. Está basado en SQL-92, proporcionando un superconjunto de la sentencia SELECT. OQL no posee primitivas para modificar el estado de los objetos, ya que éstas se deben realizar a través de los métodos que dichos objetos poseen. La sintaxis básica de OQL es una estructura SELECT...FROM...WHERE..., como en SQL.

1.- Introducción a las Bases de datos Objeto-Relacionales

Una Base de Datos Objeto Relacional (BDOR) es una base de datos que desde el modelo relacional evoluciona hacia una base de datos más extensa y compleja incorporando para obtener este fin, conceptos del modelo orientado a objetos. Podemos decir que un Sistema de Gestión Objeto-Relacional (SGBDOR) contiene dos tecnologías; la tecnología relacional y la tecnología de objetos.

En una base de datos objeto-relacional se siguen almacenando tuplas, aunque la estructura de las tuplas no está restringida a contener escalares (tipos compuestos como vectores, conjuntos, etc.) sino que las relaciones pueden ser definidas en función de otras, que es lo que denominamos herencia directa.

2.- Características de las Bases de datos Objeto-Relacionales

Con las Bases de Datos Objeto-Relacional, se pueden crear nuevos tipos de datos, que permiten gestionar aplicaciones más complejas con una gran riqueza de dominios. Estos pueden ser tipos compuestos, lo que implica que se debe definir al menos dos métodos transformadores:

- Uno para convertir el tipo nuevo a ASCII
- Otro que convierte de ASCII al nuevo tipo.

Se soportan tipos complejos como: registros, conjuntos, referencias, listas, pilas, colas y arreglos.

Se pueden crear funciones que tengan un código en algún lenguaje de programación como por ejemplo: SQL, Java, C, etc.

Existe una mayor capacidad expresiva para los conceptos y asociaciones.

Se pueden crear operadores asignándole un nombre y existencia de nuevas consultas con mayor capacidad consultiva.

Se soporta el encadenamiento dinámico y herencia en los tipos tupla o registro.

Se pueden compartir varias bibliotecas de clases ya existentes, esto es lo que conocemos como reusabilidad.

Posibilidad de incluir el chequeo de las reglas de integridad referencial a través de los triggers.

Soporte adicional para seguridad y activación de la versión cliente-servidor.

Anotar como punto final de este apartado, el inconveniente que tienen las BDOR, y es que aumenta la complejidad del sistema y por tanto se ocasiona un aumento del coste asociado.

3.- Tipos de Datos definidos por el Usuario

Los usuarios pueden definir sus propios tipos de datos, a partir de los tipos básicos provistos por el sistema o por otros tipos de datos predefinidos anteriormente por el usuario. Estos tipos de datos pueden pertenecer a dos categorías distintas:

- Los tipos de objetos (*object types*)
- Los tipos para colecciones (*collection types*).

3.1 Tipos de objetos

Un tipo de objetos define a lo que conocemos como una entidad del mundo real. Se los pueden interpretar como una plantilla de objetos de ese tipo. Está compuesto por los siguientes elementos:

- Para identificar el tipo de objetos se utiliza un nombre.
- Unos atributos que pueden ser de un tipo de datos básico o de un tipo definido por el usuario, que representan la estructura y los valores de los datos de ese tipo.
- Unos métodos que son procedimientos o funciones escritos en el lenguaje PL/SQL almacenándose de esta forma en la base de datos, o escritos en C almacenándose externamente.

3.2 Métodos

A la vez que creamos un tipo de objeto, realizamos la especificación de los métodos. Los métodos se pueden ejecutar sobre los objetos de su mismo tipo. A continuación mostramos un ejemplo: si x es una variable PL/SQL que almacena objetos del tipo Alumnos_T, entonces x.FechaNacimiento() calcula la fecha de nacimiento del alumno almacenado en x.

3.2.1 Métodos constructores de tipo

Todos los tipos de objetos tienen asociado por defecto un método que se encarga de construir nuevos objetos de ese. El nombre del método es el mismo que el nombre del tipo, y sus parámetros que tenemos en dicho método son los atributos del tipo de objetos.

3.2.2 Métodos de comparación

Estos métodos son utilizados para que se pueda comparar los objetos de un cierto tipo. Esta acción se lleva a cabo indicando cuál es el criterio de comparación. Para poder hacer posible la realización de una comparación es necesario escoger entre un método MAP o un método ORDER:

- Un método de MAP es utilizado para indicar cuál de los atributos del tipo se va a utilizar para ordenar los objetos del tipo.

- Un método ORDER utiliza los atributos del objeto sobre el que se ejecuta para realizar un cálculo y compararlo con otro objeto del mismo tipo que toma como argumento de entrada. Este método debe devolver un valor negativo si el primero es mayor que el segundo, un valor positivo si ocurre lo contrario y un cero si ambos son iguales.

3.3 Tablas de objetos

Tras definir los tipos de objetos, éstos pueden utilizarse para definir otros tipos, tablas que almacenen objetos de esos tipos, o para definir el tipo de los atributos de una tabla. Por ejemplo podemos definir una tabla que utiliza un tipo de datos complejo para una de sus columnas. Una tabla de objetos es una clase especial de tabla que almacena un objeto en cada fila.

3.4 Referencias entre objetos

Los identificadores únicos, permiten que puedan ser referenciados desde los atributos de otros objetos o desde las columnas de tablas. El tipo de datos proporcionado por Oracle para soportar esta facilidad se denomina REF. Un atributo de tipo REF almacena una referencia a un objeto del tipo definido, e implementa una relación de asociación entre los dos tipos de objetos.

3.5 Tipos para colecciones

Los tipos para colecciones se definen para poder implementar relaciones 1:N. Un dato de tipo colección está formado por un número indefinido de elementos, todos del mismo tipo. Así es posible almacenar un conjunto de tuplas en un único atributo, en forma de array o de tabla anidada.

Los tipos para colecciones también tienen por defecto unas funciones constructoras de colecciones cuyo nombre coincide con el del tipo. Los argumentos de entrada de estas funciones son el conjunto de elementos que forman la colección separados por paréntesis.

3.5.1 El tipo VARRAY

Un array es un conjunto ordenado de elementos del mismo tipo. Cada elemento tiene asociado un índice que indica su posición dentro del array.

Un tipo VARRAY se puede utilizar para:

- Definir el tipo de datos de una columna de una tabla relacional.
- Definir el tipo de datos de un atributo de un tipo de objetos.
- Para definir una variable PL/SQL, un parámetro, o el tipo que devuelve una función.

No se produce ninguna reserva de espacio al declarar un tipo VARRAY. Se almacenará con el resto de columnas de su tabla, si el espacio que requiere lo permite. En caso contrario, se almacenará aparte de la tabla como un BLOB.

Es imposible poner condiciones sobre los elementos almacenados dentro de un VARRAY, en las consultas. Esta es la principal limitación que tiene este tipo de dato.

3.5.2 Tablas anidadas

Una tabla anidada es un conjunto de elementos del mismo tipo en el que no existe un orden predefinido. Estas tablas solamente pueden tener una columna que puede ser de un tipo de datos básico, o de un tipo de objetos definido por el usuario.

4.- Herencia de tipos

La herencia de tipos surge por la posibilidad de definir tipos que sean subtipos de otros supertipos. Aparte de que los subtipos definen sus propios atributos y sus métodos, los subtipos heredan los atributos y los métodos definidos para sus supertipos. Los subtipos son capaces de redefinir los métodos que heredan, que es lo que conocemos como polimorfismo.

Por ejemplo, desde el objeto general supertipo TIPO_PERSONA podemos definir el subtipo TIPO_EMPLEADO que heredarán las características de su supertipo TIPO_PERSONA. El tipo objeto especializado TIPO_EMPLEADO pueden tener atributos adicionales o pueden redefinir métodos del objeto padre TIPO_PERSONA extendiendo la funcionalidad de sus tipos objeto.

CREATE TYPE TIPO_PERSONA AS Object

(nombre varchar2(25),

fecha_nac date

member function EDAD() **return number**,

member function PRINTME() **return varchar2) not final;**

CREATE TYPE TIPO_EMPLEADO UNDER TIPO_PERSONA

(sueldo number,

member function PAGA() **return number**,

overriding member function PRINTME() **return varchar2);**

5.- POSTGRES

Comenzó como un proyecto denominado *Ingres* en la Universidad Berkeley de California. En 1986 se continuó con el desarrollo del código de *Ingres* para crear el primer SGBD objeto-relacional. En este sistema se mejora el modelo relacional a partir de que los usuarios pueden crear sus propios tipos. El usuario puede introducir en las relaciones atributos de tipo básico o predefinidos por el mismo.

POSTGRES también mantiene la propiedad de herencia entre tipos de objetos de las bases de datos objeto-relacionales. Desarrollando de una manera más rápida y compleja estos tipos para su desarrollo.

6.- SQL 99

Se trata de un SGBD objeto-relacional. SQL:99 permite crear los tipos estructurados definidos por el usuario; Estos tipos estructurados tienen un número de características:

- Pueden ser definidos para tener uno o más atributos, los cuales pueden ser:
 - Tipos empotrados como INTEGER
 - Tipos de colección como ARRAY, u otro tipo de estructuras.
- Todos los aspectos de su comportamiento son provistos mediante métodos, funciones y procedimientos.
- Sus atributos son encapsulados mediante el uso del sistema generador observador y mutador de funciones (funciones get y set).
- Las comparaciones de sus valores son únicamente realizadas mediante funciones definidas por el usuario.
- Existe las jerarquías de tipo, en las cuales más tipos especializados (subtipos) tienen todos sus atributos y todos los métodos de los tipos generalizados (supertipos), pero pueden agregar nuevos atributos y métodos particulares.

Bibliografía:

- ❑ **Sistemas de bases de datos orientadas a objetos: Conceptos y arquitecturas.** Editorial: Addison-Wesley / Diaz de Santos. Autores: Elisa Bertino, Lorenzo Martino.
- ❑ **Sistemas de bases de datos: Un enfoque práctico para diseño, implementación y gestión.** 4ª Edición. Editorial: Pearson Addison- Wesley. Autores: Thomas M. Connolly, Carolyn E. Begg.
- ❑ **Fundamentos de Bases de datos.** 5ª Edición. Editorial: McGraw Hill. Autores: Silberschatz, Korth, Sudarshan
- ❑ **Bases de Datos Orientadas a Objeto y el estándar ODMG.** Autores: Clara Martín Sastre y Enrique Medarde Caballero. <http://tejo.usal.es/~fgarcia/docencia/poo/02-03/trabajos/S1T3.pdf>
- ❑ <http://kybele.escet.urjc.es/documentos/BD/T3-ModeloOR.pdf>
- ❑ http://informatica.uv.es/iguia/DBD/Practicas/boletin_1.pdf
- ❑ http://informatica.uv.es/iguia/DBD/Teoria/capitulo_4.pdf
- ❑ **ODMG en la Wikipedia** . <http://en.wikipedia.org/wiki/ODMG>

Tiempos necesarios para la realización:

- ❑ **Búsqueda de información:** 3 horas
- ❑ **Análisis de la información obtenida:** 10 horas
- ❑ **Selección de contenidos:** 3 horas
- ❑ **Generación del Documento:** 9 horas